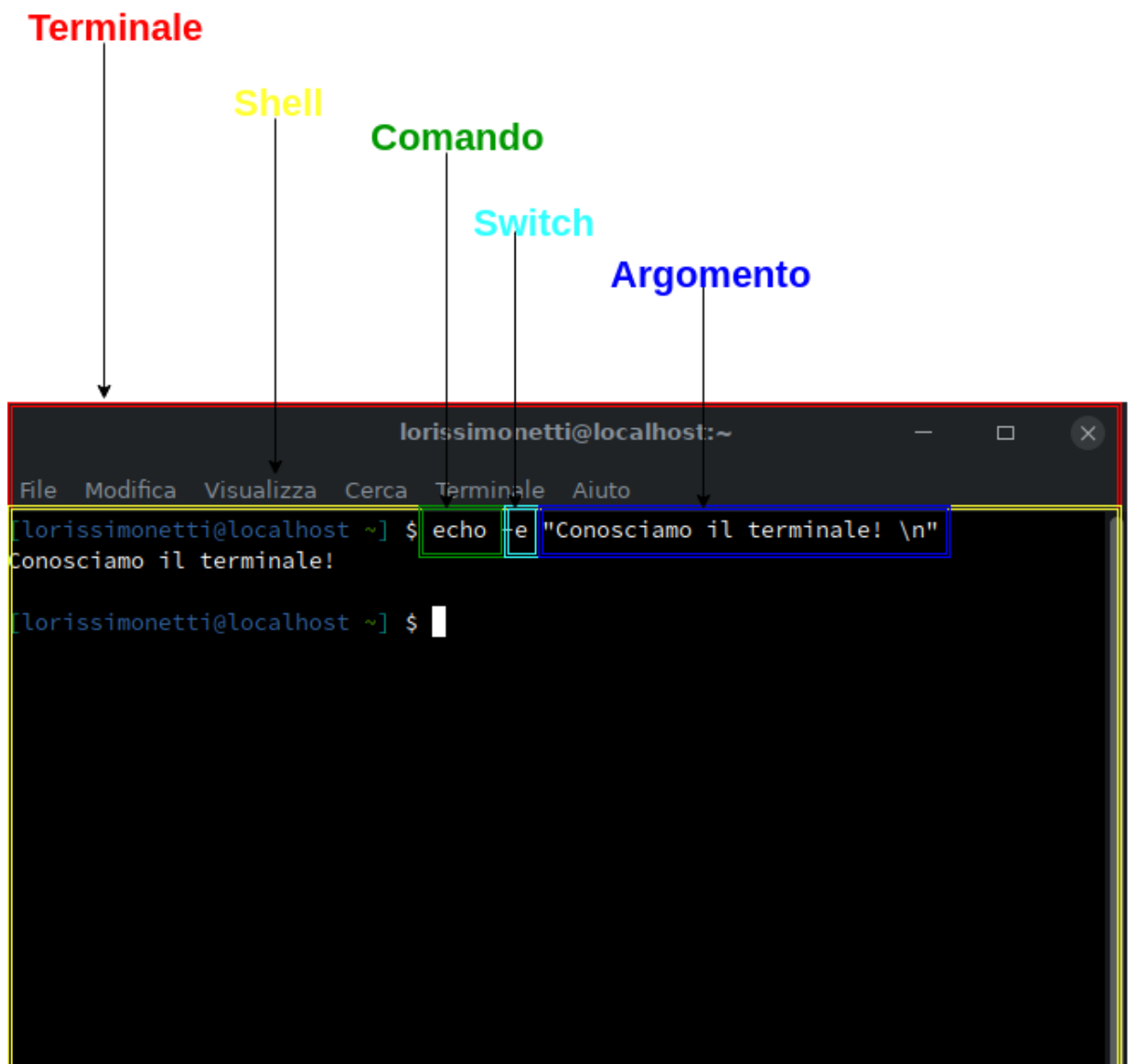
5. Iniziamo! Il terminale

Chi ha utilizzato almeno una volta sistemi unix-like avrà avuto a che fare, (o ha visto almeno una volta) con il terminale, per chi invece crede di non averne mai sentito parlare si sbaglierà, infatti pur con estrema esaltazione lo avrà già visto in moltissimi film.

Immagine tratta da Wargames - Giochi di guerra

si tratta di quella 'cosa' che fa apparire tante scritte casuali quando "l'hacker" sbatte cose a caso sulla tastiera, chiaramente nella realtà è un cpò diverso (e purtroppo più noioso).
Attraverso la riga di comando è possibile eseguire operazioni molto efficaci, che in certi casi risultano quasi impossibili utilizzando l'interfaccia grafica.
A seconda della distro e dell'ambiente desktop utilizzato, il terminale *virtuale*[5] si può presentare in differenti modi, tuttavia sono tutti pressoché uguali, tanto ciò che veramente importa è la shell, se non sapete cos'è lo vediamo subito osservando la composizione del nostro terminale.

[5] Un terminale virtuale (o emulatore di terminale), in informatica, è un programma o un servizio del sistema operativo che emula il comportamento di un terminale testuale. È notevolmente utilizzato nelle distribuzioni GNU/Linux.

- Terminale
  - Brevemente non è altro che il programma che fa girare al suo interno una shell.
- Shell
  - Detta anche interprete dei comandi, è parte integrante del sistema operativo (anche Windows ne ha una, il cosiddetto prompt dei comandi), rappresenta il box nero del terminale, dove possiamo lanciare comandi che eseguono altri programmi e ci consentono di interagire così con il sistema operativo stesso.
- Comando

- I comandi che possiamo lanciare possono riguardare un programma o un gruppo programmi, in questo caso è stato lanciato il comando “echo”, che non fa altro che stampare la riga di testo che gli viene fornita in input.
- Switch
    - Pensiamo ad esempio ad uno switch ferroviario, a seconda se è impostato o meno, cambia la direzione del treno, in questo caso il “-e” rappresenta lo switch che abilita l’interpretazione per i backslash, se fosse stato omesso, avremmo avuto in output il messaggio “Conosciamo il terminale! \n” senza alcun ritorno a capo.
      In breve lo switch cambia il sentiero da seguire, a volte consente di prendere anche degli input (è un opzione).
- Argomento
    - Nient’altri che valore che forniamo, in questo caso la frase "Conosciamo il terminale! \n".

6. I comandi linux

Combinare gli switch:

Se abbiamo bisogno di scrivere più switch per ogni comando, anziché scrivere:

```
Comando -a -b -c
```

Per maggior comodità e chiarezza possiamo anche scrivere:

`Comando -abc`

Ottenendo lo stesso risultato.

I comandi essenziali:

Comando	Descrizione	Switches Comuni + Utilizzo
`ls`	Elenca informazioni su file ed il contenuto delle directory.	**-l** Produce un elenco esteso con differenti informazioni. -h (in uso con -l) Human-readable stampa le dimensioni dei file in formato facilmente leggibile come 1K 234M 2G ecc. -a Include i file (e cartelle) nascosti
pwd	(Print Working Directory) Mostra la directory in cui ci troviamo.	**-L** usa il valore della variabile d'ambiente PWD ma solo se nelle componenti del *pathname* assoluto in esso indicato non sono usate le directory speciali "." e "..". **-P** risolve eventuali collegamenti simbolici presenti nelle componenti del *pathname* assoluto.

mkdir	(Make Directory) Usato per creare una directory, può essere utilizzato per path assolute e relative.	**-m** Specifica i permessi d'accesso da attribuire alle directory create **-p** Crea anche eventuali directory intermedie esplicitate nei parametri *dir* (se non esistono già). Esempio semplice utilizzo: `mkdir /Documenti/test`
cd	(Change Directory) Usato per spostarsi nelle directory.	N.B. "." equivale alla directory corrente. Esempio (se ci troviamo nella cartella Documenti): `cd .` = Documenti ".." equivale alla directory sovrastante. Esempio (se ci troviamo nella cartella Documenti): `cd ..` = /root (~) La directory principale è "/root" a cui possiamo accedere sia

		digitando “/root” sia con la tilde “~”. Esempio semplice utilizzo: `cd Scrivania/`
cp	(Copy) Copia il file o cartella specificato nella cartella desiderata.	**-i** (Interactive) chiedi prima di sovrascrivere un file esistente. **-r** (recursive) copia la cartella, i contenuti della cartella, e i contenuti di ogni sottocartella in essa contenuta Esempio semplice utilizzo: `cp script.sh /opt`
mv	(Move) Usato per muovere o rinominare file o cartelle.	**i** (Interactive) chiedi prima di sovrascrivere un file esistente. **-v** (verbose) da in output la lista dei file che sono stati spostati o la lista dei file che sono stati rinominati Esempio semplice utilizzo: `cp script.sh /opt`

cat	(Concatenate)
	Stampa o concatena il contenuto di uno o più file. Puoi usare anche `head` o `tail` per stampare solo le prime righe (-n nrighe), o anche `less` con grandi file perchè consente lo scorrimento del testo.
		Esempio semplice utilizzo:
		`cat file1 file2 >> risultato.txt`
		
		`cat risultato.txt`

nano	Editor di testo “amichevole” da terminale, molto semplice da	**-b** (backup) quando viene salvato un file esegue un

	utilizzare, anche per chi è ancora alla prime armi, se vuoi utilizzarne uno più avanzato prova a lanciare il comando `vimtutor`.	backup della sua precedente versione. Esempio semplice utilizzo: `nano file.txt`
grep	Cerca una stringa in uno o più file e stampa tutte le righe che corrispondono alla stringa cercata.	**-i** (ignore case) non case-sensitive, ovvero non importa la differenza tra maiuscole e minuscole. **-r** (recursive) per cercare in un intera cartella/cartelle **-l** Indica solo i nomi dei file in cui è stata trovata almeno una corrispondenza, senza stampare la stringa che corrisponde al testo cercato. **-v** Inverte la corrispondenza, Seleziona solo le righe che non corrispondono. N.B. Il comando grep può essere utilizzato anche in combinazione ad un comando che fornisce output, per stampare solo le linee che ci interessano. Esempio : `ls -lah | grep "Testo.txt` Approfondiremo il pipe(|) qualche pagina più avanti.

		Esempio semplice utilizzo: `grep "ciao" testo.txt`
sort	Stampa un file in ordine alfabetico.	**-r** (reverse) stampa in ordine inverso. Esempio semplice utilizzo: `sort testo.txt`
uniq	Stampa o omette le righe ripetute.	**-d** Stampa solo i duplicati. Esempio semplice utilizzo: `uniq testo.txt`
sudo	Usato per lanciare i comandi come root (permessi di amministratore)	Esempio semplice utilizzo: `sudo nano file.txt` N.B: I comandi eseguibili utilizzando sudo sono presenti nella cartella /sbin, mentre gli altri nella cartella /bin. Digitando `sudo su` avremmo accesso come root alla shell, e non ci sarà più bisogno di dover scrivere ogni volta il comando sudo, tuttavia prestate attenzione nell'utilizzare la shell come amministratori perchè come ogni comodità può rappresentare un rischio per la sicurezza.

Tips and tricks:

- Anziché digitare per intero un comando, basterà digitare una parte del comando (es. `cd Cart`) e premendo il tasto Tab verrà automaticamente compilato il comando (es. `cd Cartella/`).
- Utilizzando le frecce direzionali possiamo "spostarci" tra i comandi già eseguiti, in modo da poterli correggere o lanciarli di nuovo senza doverli riscrivere.

- Lanciando il comando `history` possiamo vedere la cronologia di tutti i comandi che sono stati lanciati.
- Se non conosci l'utilizzo di un comando puoi lanciare `man <comando>` per accedere al manuale di utilizzo, o se vuoi giusto darti un idea di cosa si tratta puoi lanciare `whatis <comando>`.
- Se hai bisogno di un comando per fare una determinata cosa ma non conosci il nome prova `apropos <esigenza>`. ad esempio se sei in cerca di una calcolatrice da terminale ti basterà scrivere `apropos calculator`.
- Per avere sempre a portata di mano la lista dei comandi essenziali lancia `help`.

I permessi:

Gli utenti di ciascun file directory appartengono a tre classi utente: *proprietario*, *gruppo di appartenenza* e *altri.*

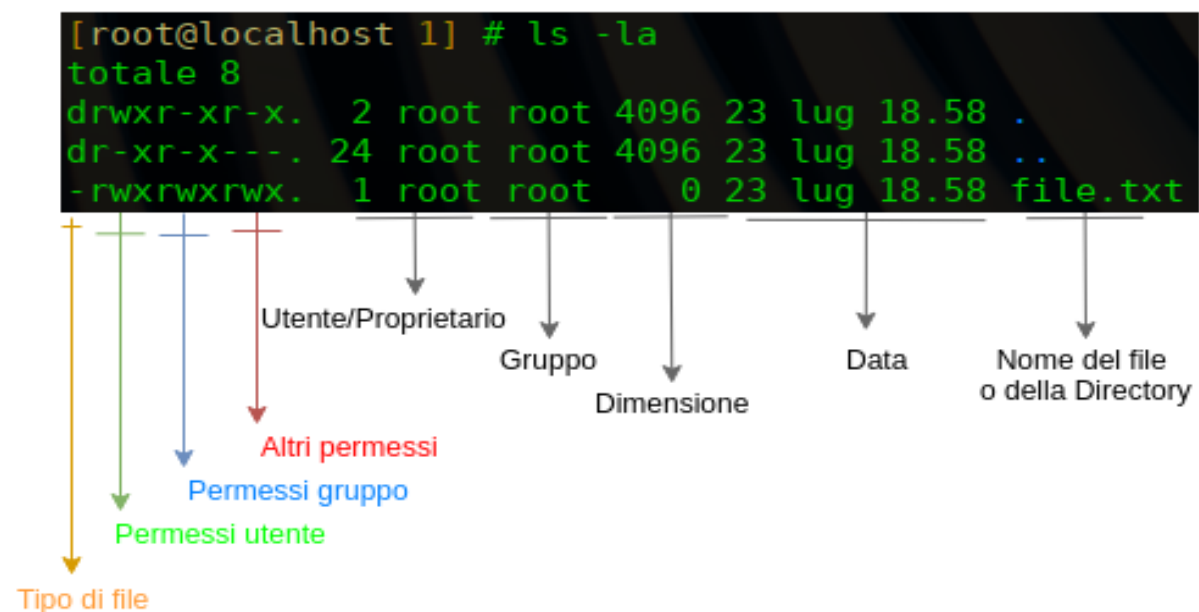
Ad ogni file o cartella è possibile associare diversi permessi di base:

- scrittura (w)
- lettura (r)

- esecuzione (x)

relativamente a ciascuna di queste tre classi di utenti:

- *proprietario* (il suo creatore). Per tale utente si applica la classe di permessi detta *utente*;
- *gruppo di appartenenza* (il gruppo *principale* del suo creatore). Gli utenti che non sono proprietari del file, ma sono membri del gruppo di appartenenza, hanno la classe di permessi detta *gruppo*. Mentre un file appartiene esattamente ad un gruppo di utenti, un utente del sistema può essere membro di uno o più gruppi (di cui uno è detto *principale* mentre gli altri sono detti *supplementari*).
- per tutti gli altri utenti che non ricadono nei due casi sopra elencati si applica, la classe di permessi detta *altri*.

La lista dei permessi legati ai file è visibile tramite il comando `ls -l`

Cambiare i permessi (chmod):

I permessi possono essere modificati con `chmod` :

```
chmod <utente(u)/gruppo(g)/altro(o)>+<permesso>+file
```

È permessa anche la modifica di più gruppi in una singola riga.

```
[root@localhost 1] # ls -l
totale 0
----------. 1 root root 0 23 lug 18.58 file.txt
[root@localhost 1] # chmod u+wrx file.txt
[root@localhost 1] # ls -l
totale 0
-rwx------. 1 root root 0 23 lug 18.58 file.txt
[root@localhost 1] # chmod g+x file.txt
[root@localhost 1] # ls -l
totale 0
-rwx--x---. 1 root root 0 23 lug 18.58 file.txt
[root@localhost 1] # chmod o+r file.txt
[root@localhost 1] # ls -l
totale 0
-rwx--xr--. 1 root root 0 23 lug 18.58 file.txt
```

È possibile impostare i permessi anche mediante riferimenti binari, la rappresentazione binaria della stringa rwx è: r=4 w=2 x=1.

Spiegazione dei permessi:

0477	-r--rwxrwx	Il proprietario può solo leggere, gruppo e altri possono fare tutto (rwx)
0677	-rw-rwxrwx	Il proprietario può solo leggere o scrivere, altri e gruppo possono fare tutto (rwx)
0444	-r--r--r--	Tutti possono solamente leggere
0666	-rw-rw-rw-	Tutti possono solamente leggere o scrivere
0400	-r--------	Il proprietario può solo leggere, gruppo e altri non possono fare nulla
0600	-rw-------	Il proprietario può solo leggere o scrivere, gruppo e altri non possono fare nulla
0470	-r--rwx---	Il proprietario può solo leggere, gruppo può fare tutto, altri non possono fare nulla
0407	-r-----rwx	Il proprietario può solo leggere, altri possono fare tutto, gruppo non può fare nulla
0670	-rw-rwx---	Il proprietario può solo leggere o scrivere, il gruppo può fare tutto, altri non possono fare nulla
0607	-rw----rwx	Il proprietario può solo leggere e scrivere, il gruppo non può fare nulla, altri possono fare tutto

0777	rwxrwxrwx	Nessuna restrizione sui permessi. Chiunque potrebbe fare tutti. Generalmente non è un impostazione consigliata.

Il peggio che può accadere a seguito dell'utilizzo dei permessi 777 su una cartella o su un file, è che se un criminale è grado di caricare un file malevolo o modificare un file corrente per eseguire il codice, avrà il pieno controllo della macchina.

setuid/setgid:

setuid (set user id) e setgid (set group id) sono permessi speciali attribuibili a file e directory che modificano il comportamento del sistema nei loro confronti.

Possono essere impostati o rimossi tramite il comando chmod.

- File eseguibili
  - ★ permette ad un utente che già possiede appropriati permessi di esecuzione sul file di eseguirlo con anche i privilegi dell'utente proprietario del file oltre che ai propri.
    - Il permesso *setuid* viene tipicamente usato per permettere ad utenti non privilegiati di

eseguire particolari programmi con i privilegi dell'amministratore (root).
  - Il permesso *setgid* su file eseguibili si comporta in maniera analoga a setuid, con la differenza che il kernel[6] attribuisce al nuovo processo un effective group ID pari a quello del gruppo assegnato al file, per cui il processo dispone anche dei privilegi di tale gruppo.
- File non eseguibili
  - ★ Il permesso *setgid* impostato su file non eseguibili permette ai processi di usare su di essi il *mandatory locking*[7], che è un meccanismo di lock non aggirabile dai processi.

Pipe e reindirizzamento:

"|" (pipe, ⇧ + \) Invia l'output di un comando come input per il prossimo.

```
[lorissimonetti@localhost ~] $ cat Testo.txt | head -n 3
Ciao
Mondo
Mondo
[lorissimonetti@localhost ~] $ cat Testo.txt | head -n 3 | sort -r | uniq
Mondo
Ciao
[lorissimonetti@localhost ~] $
```

">" Invia l'output su un file

```
[lorissimonetti@localhost ~] $ echo "Questo testo sovrascrive il precedente" > Testo.txt
[lorissimonetti@localhost ~] $ cat Testo.txt
Questo testo sovrascrive il precedente
[lorissimonetti@localhost ~] $
```

">>" Aggiunge l'output su un file (senza sovrascrivere)

[6] il nucleo di un sistema operativo, che gestisce le funzioni di controllo fondamentali del computer.
[7] Il *mandatory locking* (blocco obbligatorio) è il blocco dei file forzato dal kernel. https://www.kernel.org/doc/Documentation/filesystems/mandatory-locking.txt

```
[lorissimonetti@localhost ~] $ echo "Questo testo si aggiungerà al precedente" >> Testo.txt
[lorissimonetti@localhost ~] $ cat Testo.txt
Questo testo sovrascrive il precedente
Questo testo si aggiungerà al precedente
[lorissimonetti@localhost ~] $
```

I Data stream(flussi di dati):

I flussi di dati sono una sequenza di segnali coerenti codificati digitalmente utilizzati per trasmettere o ricevere informazioni che sono in fase di trasmissione, può sembrare complicato ma vi basta sapere che linux ne ha tre:

- Standard Input(stdin): 0
- Standard Output(stdout): 1 (il tuo terminale)
- Standard Error(stderr): 2

Il pipe e il reindirizzamento sono il mezzo tramite il quale connettiamo questi flussi fra programmi e files per indirizzare i dati.

Cosa significa? proviamo ad usare un comando non valido:

```
[lorissimonetti@localhost ~] $ ./nonvalido
bash: ./nonvalido: No such file or directory
```

Come è facile aspettarsi otteniamo un messaggio di errore, un Error Output, che come abbiamo visto è il numero 2.
Proviamo ad indirizzarlo ad un file differente (errors-log.txt):

```
[lorissimonetti@localhost ~] $ ./nonvalido 2>errors-log.txt
[lorissimonetti@localhost ~] $ cat errors-log.txt
bash: ./nonvalido: No such file or directory
[lorissimonetti@localhost ~] $
```

ora abbiamo l'errore che doveva stampato a video all'interno del file errors-log.txt

```
[lorissimonetti@localhost ~] $ ./nonvalido 1>errors-log.txt
bash: ./nonvalido: No such file or directory
[lorissimonetti@localhost ~] $ cat errors-log.txt
[lorissimonetti@localhost ~] $
```

Come possiamo vedere la stessa cosa non sarebbe accaduta se anziché 2 avessimo messo 1, che rappresenta invece lo standard output.